Position paper for WIFT'98

Martin S. Feather, Julia Dunphy & Nicolas Rouquette
Jet Propulsion Laboratory, California Institute of Technology
Mail Stop 125-233, 4800 Oak Grove Drive, Pasadena CA 91109-8099
Martin.S.Feather@jpl.nasa.gov, Julia.Dunphy@jpl.nasa.gov,
Nicolas Rouquette@jpl.nasa.gov

TOPIC AREAS ADDRESSED: Integrated Techniques, High Assurance Systems & Incorporating FM into Industrial Processes

BACKGROUND: Spacecraft fault protection software as a challenging and critical system.

A modern spacecraft is a complex mechanism controlled by software. The correct operation of such software can be crucial to the success of a spacecraft mission. A particularly critical area is fault protection – the mechanism that responds to faults, whether originating in hardware or software. Development of fault protection involves many experts, representing the many different aspects of spacecraft design (e.g., navigation, propulsion, power). This need to combine expertise from multiple areas, together with the critical nature of the resulting combination, poses a fruitful but challenging area for potential application of formal methods.

POSITION: "Pushbutton" analysis via integration of industrial tools with formal validation.

Our goal is to use the same specifications for the starting point of both the code development (whether manual or automated), and the formal analysis. Furthermore, we aim to automate as much as possible of the analysis process, to make it an easy-to-apply, "pushbutton" like activity. Towards this end, we are integrating an industrial tool (Rational Rose, a UML tool) with a formal methods tool (SPIN, a model checker developed by Gerard Holzmann of AT&T). In particular, we are developing an automatic translator from Rational Rose Statecharts to Promela (the input language of SPIN). This is inspired by the work of Erich Mikk et al., being reported in this same workshop ("Implementing Statecharts in Promela/SPIN" Erich Mikk, Yassine Lakhnech, Michael Siegel, Gerard J. Holzmann). We intend to use this translator to convert statechart descriptions of fault protection designs into Promela, and apply SPIN to check crucial properties of those designs.

RATIONALE: Our reasons for taking this approach.

- acceptance of statechart notation: our experience suggests that statecharts are readily adopted. Experts from multiple areas are willing and able to use them as an agreed-upon notation for expression and transfer of knowledge.
- rigor and expressivity of statechart notation: we have seen statecharts used to describe a spacecraft's fault protection mechanism. These specifications are rigorous and detailed enough to serve as the starting point for both development of the fault protection software, and validation of crucial properties of the design.
- good match with a fruitful analysis method: finite-state exploration techniques (some flavors of which are called "model checking") are well-suited to validating properties of moderately complex concurrent state machines. Our colleagues have successfully followed this approach on fault-protection software [1].
- automatability of approach: we are confident that the whole approach can be highly automated. The translation from statecharts into Promela (SPIN's input language) appears automatable; SPIN itself is automatic; the results from analysis are easy to render in a comprehendable manner.

CHALLENGES AND IMPLICATIONS: Challenges we have encountered, and their implications.

Challenges:

- subset of statecharts: the full extent of statechart notation is challenging to understand and handle. Fortunately, our specific application (fault protection specifications expressed in statecharts) employs only a subset of statechart notation. Indeed, the spacecraft experts deliberately restrict themselves to a notational subset so as to emerge with design documentation that is clearly and easily comprehended. Guided by these needs, we are developing the translation tool to first handle the subset of statechart notations used in that application. For example, our application's statecharts do not make use of concurrent substates (although there are multiple statecharts, corresponding to concurrent state machines), so we have not yet addressed this statechart capability. In general, we expect that our translation tool can be smoothly extended to handle these omitted capabilities.
- superset of statecharts: the authors of the statecharts we wish to handle have used some mutually-agreed upon conventions which extend the standard semantics of statecharts. Again, driven by the needs of our specific application, we are incorporating these extensions into our translator.

Implications:

- current capabilities of our translator: To date, the major capabilities of our translator tool are: multiple statecharts; nested states; guard conditions, events and actions on transitions; actions on entry into and exit from states; simple "do" actions in states (corresponding to the invocation of an entire statechart). The major omissions are concurrent substates, history, real time; we require actions be written in Promela (e.g., variable assignments, queue operations).
- structure the translator tool: we would like to structure our translator tool itself so as to make it readily extensible and customizable. We are viewing our translator tool as an instance of a code generator tool (even though it generates something intended for analysis, not execution). We have ideas on how to organize code generation as a dataflow-like process, which will, we hope, give it the flexibility we desire. How do other people address this issue?
- sharing of translation capabilities: it would be nice to be able to make use of pieces of other people's translation tools. For example, if Erich Mikk's MOCES tool handles translation of concurrent substates, could we incorporate that portion of his tool? Sharing actual code is perhaps too much to hope for, but we would certainly hope to be able to share approaches and solutions. At the very least, we would like to know what other efforts along these lines are underway.
- need for automated abstraction: it is probable that we will need some automated abstraction to help reduce the complexity of the finite state machines to tractable levels. We are aware of some work towards automated abstraction as a prelude to model checking (e.g., [2]). Again, we would like to know what work is going on in this area, and, ideally, find a way to make use of the emerging results.

---

[1] F. Schneider, S.M. Easterbrook, J.R. Callahan & G.J.Holzmann, "Validating Requirements for Fault Tolerant Systems using Model Checking." *International Conference on Requirements Engineering*, 1998.

[2] C. Heitmeyer, J. Kirby & B. Labaw, "Applying the SCR Requirements Method to a Weapons Control Panel: An Experience Report." *2nd Workshop on Formal Methods in Software Practice*, 1998.

ACKNOWLEDGMENT: The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.